Project 2:

Title:
Mastermind AI

Class:
CSC 7 - 44265

Due Date:
May 29th, 2022

Author of Driver/Template:
Dr. Mark E. Lehr

Author of AI Function:
Aamir Khan

Introduction and Gameplay:

Title: Mastermind AI

Objective:

Mastermind is a code-breaking game that's usually played between two players. The objective of Mastermind is to break a code or correctly guess a random code that is either randomly generated or chosen by the other player. If a player can manage to break the code generated within a specific number of attempts, the player wins. Otherwise, the game ends and no winner is declared. To make the game more difficult for players, the option of using duplicate digits in the random code generated is available. This complicates the game because now more options can be tested or guessed as a result of allowing duplicates to be used.

The "How-To":

Each round consists of making a guess to break the code. After each attempt, hints will be displayed to assist the player in breaking the random generated code. An "rr" indicates that a digit is right and in the right place. However, the player won't know exactly what digit is correct or in the correct place. An 'rw' indicates that a digit is right but in the wrong place. This means that 1 of the digits is correct, but that it needs to be placed in a different position.

Overview:

For this project, the driver was provided to me, and I had to write the AI function. This AI function would make guesses for the code to break and essentially attempt to crack the code. In other words, the program would have to crack the code itself using the hints that would be provided for the guesses made by the AI function. I utilized static variables to retain the historical information of my variables. This way, the AI would keep track of the information provided by the previous guesses. I went with a basic approach: find all of the correct digits first, find the position of the 1st digit, find the position of the second digit, and find the position of the third digit and fourth digits. This approach would take more than 10 guesses, but the goal of this project was for the AI to be able to crack the code as fast or even faster than a binary search. The code itself wasn't too challenging to write, but most of my time was spent working out the logic behind my AI and how it would utilize the hints provided to correctly save important information, such as the correct digits in our code to break and the positions of our digits.

Stages of Development:

Version 1: Finding All Correct Digits

In this version, the objective was to find all of the correct digits within our code to break. The idea was to go through the guesses 0000-9999 and determine which guesses returned an “rr” hint, which indicates that we have a correct digit. This is the most basic approach to determine which four digits are in our code. This would take at most nine guesses, so we’d have all of our digits by the tenth guess. At this point, the AI is only capable of finding the four digits in the code that we are expected to break. In the next version, I take care of finding the position of the first digit in our code. This would mean that only one of the digits will be in its correct position.

Version 2: Finding Position of 1st Digit

In this version, I deal with finding the position of the first of our four digits. I begin by trying the digit in the first position. If the digit is in the wrong position, I try the digit in the second position on the next guess. If the digit is still in the wrong position, I try it out in the third position. If this also happens to be the wrong position, then we know that the only remaining position must be our digit’s correct placement. Once I find the correct position, I save the position of our first digit. Finally, I return the guess with the correct first digit. For the sake of knowing which digit that is, all other digits in our guess will be digits that aren’t found in our code to break.

Version 3: Finding Position of 2nd Digit

Much like version 2, I deal with finding the position of the second digit in our code. Just as we did in version 2, I try out the second digit in various positions until I find the correct position. I begin by placing the first digit in its correct place. I then place the second digit in the first position I can try it in. I set all other digits as wrong digits. This way, we can meaningfully use the “rr” and “rw” hints. If the other digits weren’t set to wrong digits, then the “rr” and “rw” hints could possibly be misleading in determining the positions of our digits. If the second digit is in the wrong place, I try it out in the second available position. If that happens to be the wrong position, the only other remaining position is our correct placement. I save the position once I find it and return a guess with the first two correct digits. All other digits are set to wrong digits.

Version 4: Finding Position of 3rd Digit

Just as we did in versions 2 and 3, we follow much of the same logic. The only difference is that we now have less steps to take care of because there are only 2 positions left that I need to find. I find the positions that I can try the third digit in. I begin by trying out the first position that the third digit can be in. If this happens to be the wrong position, then the only other remaining position has to be the correct placement of our third digit. I save the third position and return a guess with three of our digits correct. The fourth digit is set to a wrong digit, but I'll correctly set that last digit in the final version of my program. At this point, the AI is now capable of finding the correct positions of three digits.

Version 5: Cracking the Code

The only step we have left is to set the position of our fourth digit. Since we know the positions of three of our digits, the only remaining position has to be the position of the fourth digit. I save that last position as the position of our fourth digit. I can now finally return a guess with all four digits in their correct positions. Moreover, the AI is now capable of cracking the code in less than or equal to 16 guesses. This is what we were aiming for, so now the program is complete.

Mastermind AI Pseudo-Code:

```
/*
 * File:   main.cpp
 * Author: Aamir Khan
 * Created on May 19, 2022, 10:30 AM
 * Purpose: Mastermind AI Pseudo-Code
 */


//System Libraries
//I/O Library
//Cstd. Library
//Ctime Library
//Cstring Library
//Namespace Std of System Libraries

//Function Prototypes
//List my function prototypes.

//Main -> Execution Begins Here:

        //Initialize the Random # Generator Seed.

        //Declare My Variables and Initialize Them.
        //Declare and Set the Code to Break.

        //Looping through guesses until a correct guess is returned.

                //Guessing from 0000, 1111, …, 8888 or until all 4 digits in our code are found.
                //If by guess 8888, we don't have all of our digits, then 9 has to be our last digit.
                //Once we find all 4 digits, we begin looking for the positions of our digits.
                        //Find the position of the 1st digit in numerical order.
                        //Find the position of the 2nd digit in numerical order.
                        //Find the position of the 3rd digit in numerical order.
                        //The position of the 4th digit is simply the remaining spot.
                //A correct guess will break out of our do-while loop.

        //Output how many guesses it took to solve the game.
        //Display the code that was to be broken alongside our final guess.

        //Quit the Program.

//Exit main - End of the Program
```

Sample Screen Shots:

```
string sGuess="0000";   //To hold our guess string.
static int digit=-1;    //To hold which digit we're checking to see is in our code (0-8).
static int nGuesses=1;  //To hold how many guesses we've made.
static int nClrs=0;     //# of correct digits/colors we've found (Should total to 4).
static int index=0;     //Index variable for our arrays.
static int nWaysA=1;    //To hold the current way we're trying out the 1st digit.
static int nWaysB=1;    //To hold which way we're trying out the 2nd digit or B.
static int nWaysC=1;    //To hold which way we're trying out the 3rd digit or C.
static char clrs[5];    //To hold our correct colors + null term.
clrs[4]='\0';           //Setting the null term.
static bool fndPos1,    //Whether we've found our first digit's position.
            fndPos2,    //Whether we've found our second digit's position.
            fndPos3;    //Whether we've found our third digit's position.
if(nWaysA==1&&!fndPos1) //If we're on our 1st way to place A, and we haven't already found pos1,
    fndPos1=false;      //then we set it to false. (Accounts for when our digit is in index 0).
if(nWaysB==1&&!fndPos2) //If we're on our 1st way to place B, and we haven't already found pos2,
    fndPos2=false;      //then we set it to false. (Accounts for when our digit is in index 0).
if(nWaysC==1)           //If we're on our 1st way to find C, then we set fndPos3 to false.
    fndPos3=false;      //Only on the 1st way, we set it to false.
static int pos1=-1,     //Correct index of the 1st correct digit.
           pos2=-1,     //Correct index of the 2nd correct digit.
           pos3=-1,     //Correct index of the 3rd correct digit.
           pos4=-1;     //Correct index of the 4th correct digit.
static char wrong;      //To hold a wrong digit. (We'll use this to find our positions.)
static bool isVldHnt;   //To hold whether we have a valid hint or not.
                        //When we make the guesses 0000-8888, if one of those digits happens
                        //to be in our code, rr would = 1 on the next guess, and this could
                        //falsely set fndPos1 to true. We want to set fndPos1 only when we've
                        //found all of our digits and the hint is not for guesses 0000-8888.
static int foundOn;     //To hold what guess number we found a position on.
```

All our variables utilized by the AI function - Most of these variables are static variables.

```
if(digit!=8){   //If we haven't tried digits 0-8...
    digit+=1;   //0,1,2,...,8.
}
//Store our guess as a string. All digits are the same (0000-8888):
sGuess[0]=sGuess[1]=
sGuess[2]=sGuess[3]=digit+'0'; //We need to add '0' because digit is an int.
```

Trying out guesses with the same 4 digits - This is when we want to find our 4 correct digits.

```
//If we have a correct digit, we save it into our colors array.
//We know that a color is correct if rr is >=1 & <= 4.
//We also can't already have 4 digits by this point.
else if(rr>=1&&rr<=4&&nClrs!=4){
    //Copy over the colors that are correct. We can have duplicates.
    for(int i=index;i<index+rr;i++){
        //Because the rr hint is based off of the previous guess,
        //we need to subtract the digit we're currently testing by 1.
        //For ex. if we guessed the digit 7 (7777), then we'd know by the next guess if it is
        //in our code. At this point, we'd be testing 8 (8888), so we subtract 8 by 1.
        clrs[i]=(digit-1)+'0'; //The digit that produced the rr hint.
        nClrs++;               //We've found a color/digit, so we increment.
    }
    index+=rr;                 //We increment by rr to go to the next available index.
}
```

If our “rr” hint happens to be set, we store the digit the hint is based on as a correct digit.

```
//If we have a correct digit in the wrong place, we try out the other ways...
//(W A W W), (W W A W), & (W W W A).
if(rw==1){
    nWaysA++; //Trying out a different positioning or way.
    //We set the next peg as the correct color.
    sGuess[++index]=clrs[0];
    //Again, the rest become wrong digits.
    for(int i=0;i<4;i++){
        if(i!=index)sGuess[i]=wrong;
    }
}
```

If we need to try out the first digit in another position, we try the next available position.

```
//If we've tried out 3 ways to put A (Ex. AWWW, WAWW, or WWAW) and all were wrong,
//then that means it has to be the only other way to try A (WWWA).
//This would be our fourth variation to place A in.
else if(nWaysA==4){
    fndPos1=true;
    pos1=index; //Index would be = to 3 here (Last index to try A in).
    //Store what guess # we found position 1 on.
    foundOn=nGuesses;
}
//If our code ends up being 0000, 1111, ..., or 8888, then we just return it.
if(rr==4){
    return sGuess;
}
```

If we’ve tried out three different ways to place the first digit and they were all wrong, then the only other remaining position is the correct position. (“A” represents our first digit.)

```
//If we've tried out 2 ways to put B (Ex. ABWW or AWBW) and both were wrong,
//then that means it has to be the only other way to try B (AWWB).
//This would be our third variation to place B in.
else if(nWaysB==3){
    fndPos2=true;
    pos2=bIdx[index]; //The last index B can be in (2).
    //Store what guess # we found position 2 on.
    foundOn=nGuesses;
}
```

If we've tried out 2 different ways to place the second digit and they were both wrong, then the only other remaining position is the correct position. ("B" represents our first digit.)

```
//If we tried out 1 way to put C (Ex. ABCW) and it was wrong,
//then that means it has to be the only other way to try C (ABWC).
//This would be our second variation to place C in.
else if(nWaysC==2){
    fndPos3=true;
    pos3=cIdx[index]; //Index would be = to 1 here.
}
```

If we've tried out 1 way to place the third digit and it was wrong, then the only other remaining position is the correct position of our third digit. ("C" represents our first digit.)

```
//If we've found our 3 positions, we can now return our final guess.
//The 4th position is simply the last position that's available.
if(fndPos1&&fndPos2&&fndPos3){
    sGuess[pos1]=clrs[0]; //1st correct digit.
    sGuess[pos2]=clrs[1]; //2nd correct digit.
    sGuess[pos3]=clrs[2]; //3rd correct digit.
    //Finding the position of the 4th digit. (The only remaining position).
    for(int i=0;i<4;i++){
        if(i!=pos1&&i!=pos2&&i!=pos3)
            pos4=i;
    }
    sGuess[pos4]=clrs[3]; //4th correct digit.
}
```

Once we've found the positions of three of our digits, then the last available position is our fourth digit's position. We finally set all four of our digits in their correct positions.

```
int main(int argc, char** argv) {
    //Set the random number seed
    srand(static_cast<unsigned int>(time(0)));

    //Declare variables
    string code,guess;  //code to break, and current guess
    char rr,rw;         //right digit in right place vs. wrong place
    int nGuess;         //number of guesses

    //Initialize Values
    nGuess=0;
    code=set();
    rr=rw=0;

    //Loop until solved and count to find solution
    do{
       nGuess++;
       guess=AI(rr,rw);
    }while(eval(code,guess,rr,rw));

    cout<<"Guesses to break code = "<<nGuess<<endl;
    cout<<"Code  = "<<code<<endl;
    cout<<"Guess = "<<guess<<endl;
    //Exit the program
    return 0;
}
```

Our Program's Main - we loop through as many guesses as needed to crack the code. We then output how many guesses it took as well as the code we were able to break.